

A New Training Method for Feedforward Neural Networks Based on Geometric Contraction Property of Activation Functions

Petre Birtea*, Cosmin Cernăzanu–Glăvan**, Alexandru Şişu**

*Department of Mathematics, West University of Timișoara

Bd. V. Pârvan, No 4, 300223 Timișoara, România

birtea@math.uvt.ro

**Department of Computer Science, "Politehnica" University of Timișoara

Bd. V. Pârvan, No 2, 300223 Timișoara, România

cosmin.cernazanu@cs.upt.ro; sisu.eugen@gmail.com

Abstract

We propose a new training method for a feedforward neural network having the activation functions with the geometric contraction property. The method consists of constructing a new functional that is less nonlinear in comparison with the classical functional by removing the nonlinearity of the activation functions from the output layer. We validate this new method by a series of experiments that show an improved learning speed and also a better classification error.

MSC: 92B20, 68T05

Keywords: feedforward neural network, training algorithm, backpropagation, contraction, optimization

1 Introduction

From the mathematical point of view a feedforward neural network (FNN) is a sequence of mathematical operations (of type vector multiplication and non-linear activation) able to transform vectors from the input space into vectors from the output space.

A feed forward neural network with n hidden layers can be seen as a minimization problem for the following cost function:

$$E(\mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_n) := \sum_{\alpha=1}^T \|\mathbf{s}_n(\mathbf{W}_n \cdot \dots \cdot \mathbf{s}_2(\mathbf{W}_2 \cdot \mathbf{s}_1(\mathbf{W}_1 \cdot \mathbf{x}_\alpha)) \dots) - \mathbf{y}_\alpha\|^2, \quad (1.1)$$

where:

- \mathbf{W}_i represents the weights matrices between the $(i-1)^{th}$ layer and i^{th} layer
- the vectorial activation function $\mathbf{s}_i : \mathbf{I}^{d_i} \rightarrow \mathbf{J}^{d_i}$ is defined by $\mathbf{s}_i(\mathbf{x}) := (s(x_1), \dots, s(x_{d_i}))$, where $\mathbf{I}^{d_i} = \underbrace{I \times \dots \times I}_{d_i \text{--times}}$ and $\mathbf{J}^{d_i} = \underbrace{J \times \dots \times J}_{d_i \text{--times}}$ with d_i being the number of neurons on the i^{th} layer, $s : I \subseteq \mathbb{R} \rightarrow J \subseteq \mathbb{R}$ is a neuron activation function, and $\mathbf{x} = (x_1, x_2, \dots, x_{d_i})$ is the numeric codified neural network input
- \mathbf{x}_α is the α^{th} input vector of the training set
- \mathbf{y}_α is the α^{th} output vector of the training set
- T is the number of samples in the training set.

The training of a neural network means to find the weights matrices $\mathbf{W}_1, \mathbf{W}_2, \dots, \mathbf{W}_n$ that render the minimum value of the cost function E . The most common approach is to apply an iterative numerical algorithm that in the end will generate a set of approximated optimal weights matrices ($\mathbf{W}_1^{(opt)}, \dots, \mathbf{W}_n^{(opt)}$).

Usually the activation function s is a non-linear function that increases the amount and the complexity of the computations necessary to solve the above minimization problem.

This cost function can be seen as a **distance function in the Euclidean space of the weights variables**. Hence the minimization of the cost function E can be translated into the problem of minimizing the distance between the target vectors and the output vectors of the neural network.

Figure 1 presents an intuitive depiction of the linear transformation followed by the non-linear transformation (the activation function) occurring between two consecutive layers. The cost function can be seen at the end of the transformation process.

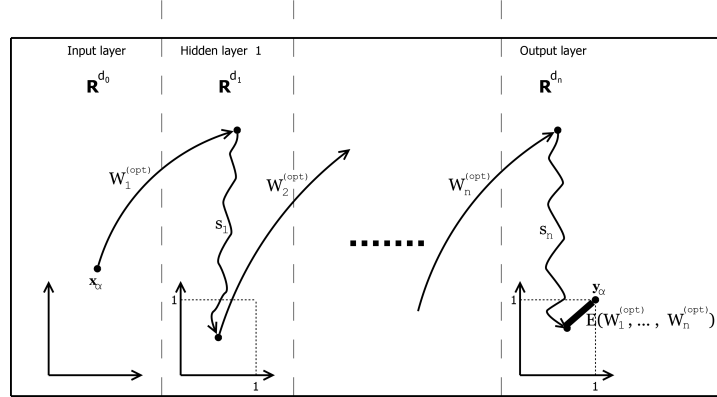


Figure 1: An intuitive geometric depiction of the activation and the error distance in a classical training scenario

Since the development of the backpropagation algorithm (BP) there was a lot of effort in order to address the shortcoming of applying the BP in really large networks. The shortcomings were vanishing gradient problem and the algorithm complexity translated into CPU time. These problems were tackled by improvements that followed several directions that we might classify into: mathematical, structural and algorithmic.

When talking about mathematical improvements we have to mention: Quasi Newton methods [3], [7], [9], [28]; conjugate gradient descent [11], [20]. However, this methods turned out to be expensive from a computational point of view. Methods like Hessian-free optimization [20], [23], [27], [19] address the problem of vanishing gradient in feedforward neural networks. Levenberg-Marquardt [15], [18], [25] improved the convergence speed.

Another set of improvements came from modifying the structure of a FNN like in the Dropout method [12], [1], [2] or by modifying the activation function from non-linear to linear like in the case of Rectified Linear Units (ReLU) [17], [21], [16], [8], [13], [4].

Numeric and algorithmic optimization techniques were also employed when trying to optimize the whole process of running BP: regularization and weight decay [10, 30, 14], momentum [24], [6], [31], Nesterov accelerated gradient descent [22]. Other gradient based optimization methods turned out to be really efficient: RmsProp [29] [26], Adagrad [5], Adadelata [33].

2 Canceling the non-linearity on the output layer

Our idea is to modify the initial cost function by taking away the non-linearity of the activation function on the output layer. This leads to reduce the amount of computation for finding the optimal values of the weights matrices that renders a minimum value of the cost function. In order to do this, we need the following hypotheses:

- the activation function s is a diffeomorphism on its domain of definition and moreover it is a contraction (i.e. the derivative of s is strictly smaller than 1 on the whole interval of definition)
- the coordinates of output vectors $\tilde{\mathbf{y}}_\alpha$ belong to the co-domain of the activation function s . The output vectors $\tilde{\mathbf{y}}_\alpha$ can be the same with the output vectors \mathbf{y}_α if the activation function s allows it, see the discussion in next section.

More precisely, we propose the following modified cost function:

$$\tilde{C}(\tilde{\mathbf{W}}_1, \tilde{\mathbf{W}}_2, \dots, \tilde{\mathbf{W}}_n) := \sum_1^T \|\tilde{\mathbf{W}}_n \cdot \mathbf{s}_{n-1}(\tilde{\mathbf{W}}_{n-1} \cdot \dots \cdot \mathbf{s}_2(\tilde{\mathbf{W}}_2 \cdot \mathbf{s}_1(\tilde{\mathbf{W}}_1 \cdot \mathbf{x}_\alpha)) \dots) - \mathbf{s}_n^{-1}(\tilde{\mathbf{y}}_\alpha)\|^2, \quad (2.1)$$

where:

- $\mathbf{s}_n^{-1} : \mathbf{J}^{d_n} \rightarrow \mathbf{I}^{d_n}$ is the inverse of the vectorial activation function \mathbf{s}_n
- $\tilde{\mathbf{W}}_i$ represents the weights matrices between the $(i-1)^{th}$ layer and i^{th} layer
- $\tilde{\mathbf{y}}_\alpha$ is the α^{th} output vector of the training set.

We propose to train the neural network in the same manner, but using the newly introduced cost function \tilde{C} . At the end, we will obtain a set of approximated optimal values $(\tilde{\mathbf{W}}_1^{(opt)}, \dots, \tilde{\mathbf{W}}_n^{(opt)})$ for the variables that we are interested, in our case the weights matrices.

We will show that introducing the computed weights $(\tilde{\mathbf{W}}_1^{(opt)}, \dots, \tilde{\mathbf{W}}_n^{(opt)})$ in the typical feedforward error function (1.1)

$$\tilde{E}(\tilde{\mathbf{W}}_1, \tilde{\mathbf{W}}_2, \dots, \tilde{\mathbf{W}}_n) := \sum_1^T \|\mathbf{s}_n(\tilde{\mathbf{W}}_n \cdot \dots \cdot \mathbf{s}_2(\tilde{\mathbf{W}}_2 \cdot \mathbf{s}_1(\tilde{\mathbf{W}}_1 \cdot \mathbf{x}_\alpha)) \dots) - \tilde{\mathbf{y}}_\alpha\|^2,$$

where the new output vectors are $\tilde{\mathbf{y}}_\alpha$, will yield even better values. Indeed, due to the fact that vectorial function \mathbf{s}_n is a contraction, this implies that it will contract the distances. Expressed in a mathematical form we have:

$$\tilde{C}(\tilde{\mathbf{W}}_1^{(opt)}, \dots, \tilde{\mathbf{W}}_n^{(opt)}) > \tilde{E}(\tilde{\mathbf{W}}_1^{(opt)}, \dots, \tilde{\mathbf{W}}_n^{(opt)}).$$

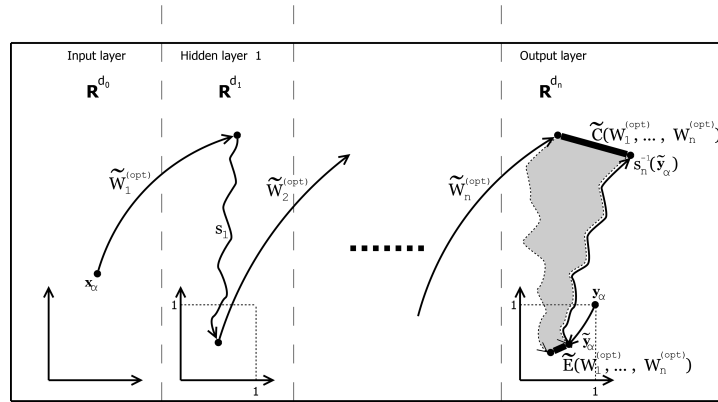


Figure 2: An intuitive geometric depiction of the activation and the error distance in the newly proposed method

Starting from an initial set of conditions and applying the standard training procedure (using the classical cost function E) we obtain a set of weight matrices $(\mathbf{W}_1^{(opt)}, \dots, \mathbf{W}_n^{(opt)})$, which in general

are different than the weights matrices ($\widetilde{\mathbf{W}}_1^{(opt)}, \dots, \widetilde{\mathbf{W}}_n^{(opt)}$) obtained by training the network using the modified cost function \widetilde{C} .

A direct comparison between the two training methods, the classical one using the cost function E and the newly proposed one using the cost function \widetilde{C} cannot be performed, as in general the inequality

$$E(\mathbf{W}_1^{(opt)}, \dots, \mathbf{W}_n^{(opt)}) > \widetilde{E}(\widetilde{\mathbf{W}}_1^{(opt)}, \dots, \widetilde{\mathbf{W}}_n^{(opt)}), \quad (2.2)$$

cannot be proved mathematically.

Experimentally, when we take as a criteria of performance the number of rightfully classified examples (accuracy measure), we prove that our newly proposed training method yields better classification results than the classical one. Also, the experiments show that as a byproduct of this method we obtain also a faster classification for the chosen datasets.

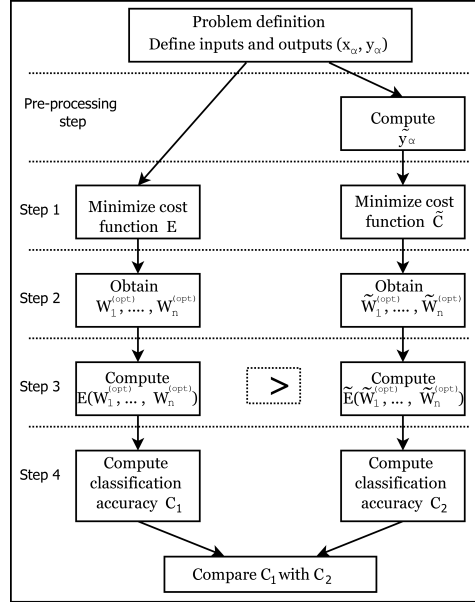


Figure 3: The logical diagram that compares the classical training method with the newly proposed one

The comparison and the implementation procedure of the classical training method and the newly proposed method follows the logical diagram depicted in Figure 3. Thus, we experimentally compare C_1 (the accuracy percentage for the classical method) with C_2 (the accuracy percentage for the newly proposed method). The newly proposed method needs a supplementary preprocessing step where we need to define the new output vectors $\widetilde{\mathbf{y}}_\alpha$. This intermediate step is necessary only when the coordinates of the output vectors \mathbf{y}_α in the classical method does not belong to the co-domain J of the activation function s . The rest of the steps remain the same with the difference that we minimize a different cost function \widetilde{C} , defined in (2.1).

Experimentally, we observe that after *Step 3* we have a faster learning (lower value of the corresponding cost functions) using the new method. This advantage is reflected at the end of the *Step 4*, where one compares the actual accuracies (C_1 and C_2) obtained by the two methods.

3 Experiments

We have done three experiments, in all three we have used the MNIST dataset [32]. This is the most common dataset (actually is a subset of a larger available set - NIST) used in this field and it contains a large set of images representing handwrite digits. The training set contains 60.000 examples and

the testing set 10.000 examples. Each experiment consists of training a neural network using the two different cost functions, the classical cost function E and respectively, the newly proposed cost function \tilde{C} . In all three experiments we have chosen neural network configurations already used in literature, see [32]. During the three experiments we have obtained a faster learning and also a better classification error using the newly proposed cost function \tilde{C} .

For the first experiment we consider a feedforward neural network having the following configuration: the input layer contains 784 neurons, followed by two hidden layers with 300, respectively 100 neurons, and an output layer containing 10 neurons (one neuron for each digit).

For the classical training of the first experiment all neurons are classical neurons having as activation function the sigmoid, $s : \mathbb{R} \rightarrow (0, 1)$, $s(x) = (1 + e^{-x})^{-1}$. Each target is a vector $\mathbf{y}_\alpha \in \mathbb{R}^{10}$ with components 0 or 1.

For the second training of the first experiment, we start by verifying that the sigmoid function obeys the first hypothesis discussed in the previous section, meaning it is a contracting diffeomorphism on its domain of definition.

In order to implement our method, the second hypothesis needs to be verified. We need to apply the inverse sigmoid on each component of the target vector \mathbf{y}_α . As " $s^{-1}(1) = \infty$ " and " $s^{-1}(0) = -\infty$ " one must replace the coordinate entries 0 and 1 in \mathbf{y}_α with other two reference values belonging to $(0, 1)$. To find a set of suitable values, a series of experiments were conducted and we obtain the best results (from the training point of view) when replacing 0 with 0.2227 and 1 with 0.7773. Thus, the coordinate entries of the new output vectors $\tilde{\mathbf{y}}_\alpha$ are 0.2227 and 0.7773. Following the logical diagram depicted in Figure 3, we implement our method accordingly:

- *Pre-processing step*: replace the vector components 0 and 1 of the target vectors \mathbf{y}_α with 0.2227, respectively 0.7773, thus obtaining the new output vectors $\tilde{\mathbf{y}}_\alpha$. Compute the vectors $\mathbf{s}_n^{-1}(\tilde{\mathbf{y}}_\alpha)$ that are used for defining the cost function \tilde{C} .
- *Step 1*: apply the gradient descent algorithm for the new cost function \tilde{C} .
- *Step 2*: after a sufficient number of iterations we obtain the weights matrices $\tilde{\mathbf{W}}_1^{(opt)}, \tilde{\mathbf{W}}_2^{(opt)}, \tilde{\mathbf{W}}_3^{(opt)}$, where $\tilde{\mathbf{W}}_1^{(opt)}$ is the weights matrix of 784×300 elements between the input layer and the first hidden layer, $\tilde{\mathbf{W}}_2^{(opt)}$ is the weights matrix of 300×100 elements between the first hidden layer and the second hidden layer and $\tilde{\mathbf{W}}_3^{(opt)}$ is the weights matrix of 300×10 elements between the second hidden layer and the output layer.
- *Step 3*: compute the value $\tilde{E}(\tilde{\mathbf{W}}_1^{(opt)}, \tilde{\mathbf{W}}_2^{(opt)}, \tilde{\mathbf{W}}_3^{(opt)})$ and compare with $E(\mathbf{W}_1^{(opt)}, \dots, \mathbf{W}_n^{(opt)})$ obtained using the classical training, see Figure 4.

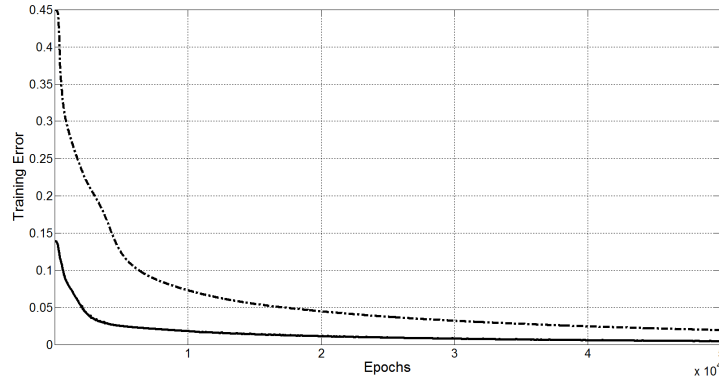


Figure 4: Training error for experiment 1. The newly proposed method (continuous line) shows a smaller error than the classical method (dash line) after 5×10^4 epochs. Also, the new method yields a faster convergence in terms of training error.

- *Step 4*: compute the accuracy value C_2 obtained with the newly proposed method and compare with accuracy value C_1 obtained using the classical training, see Figure 5. For this configuration, LeCun et al. [32] report a classification error of 3.05, value which we have obtained after 47.750 epochs with classical training. After the same number of epochs, using the newly proposed method, we have obtain a classification error of 2.59.

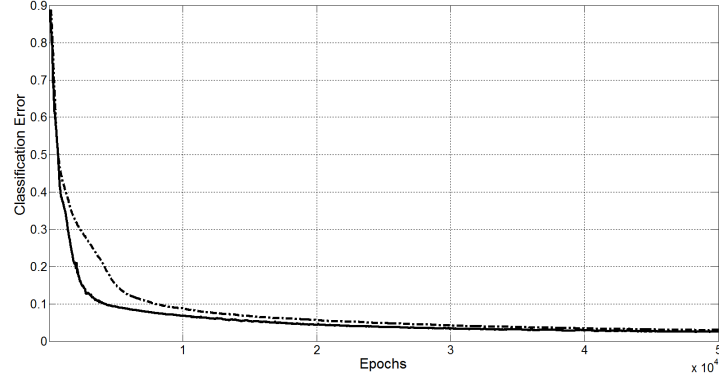


Figure 5: Classification error for experiment 1. The newly proposed method (continuous line) shows a better classification than the classical method (dash line). After 5×10^4 epochs, the classification error obtained with the new method is 2.55 vs. 3.02 which is the classification error obtained using the classical method.

For the second experiment we have changed the architecture of the neural network in the following way: the input layer contains 784 neurons, followed by two hidden layers with 500, respectively 150 neurons, and an output layer containing 10 neurons (one neuron for each digit). Both classical and the newly proposed training had followed the same procedure as in the first experiment, which we have described in details. The comparison between the two trainings of the second experiment can be seen in Figure 6 and Figure 7. For this configuration, LeCun et al. [32] report a classification error of 2.95, value which we have obtained after 77.800 epochs with classical training. After the same number of epochs, using the newly proposed method, we have obtained a classification error of 2.00.

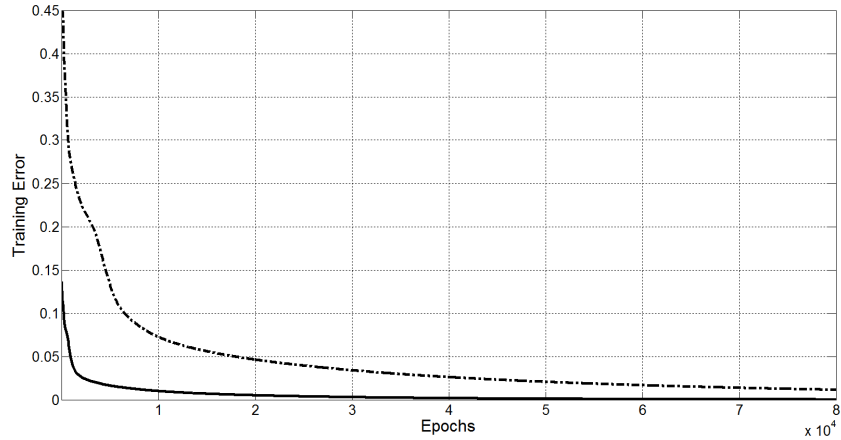


Figure 6: Training error for experiment 2. The newly proposed method (continuous line) shows a smaller error than the classical method (dash line) after 8×10^4 epochs. Also, the new method yields a faster convergence in terms of training error.

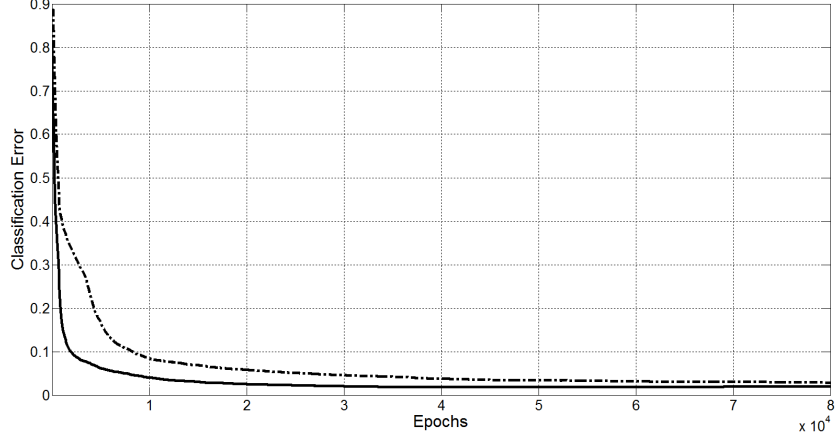


Figure 7: Classification error for experiment 2. The newly proposed method (continuous line) shows a better classification than the classical method (dash line). After 8×10^4 epochs, the classification error obtained with the new method is 1.99 vs. 2.88, which is the classification error obtained using the classical method.

In the third experiment we have considered an architecture of the neural network with a single hidden layer of 1000 neurons. Thus, the architecture is as follows: the input layer contains 784 neurons, followed by one hidden layer with 1000 neurons, and an output layer containing 10 neurons (one neuron for each digit). Both classical and the newly proposed training had followed the same procedure as in the first two experiments. The comparison between the two trainings of the third experiment can be seen in Figure 8 and Figure 9. For this configuration, LeCun et al. [32] report a classification error of 4.5, value which we have obtained after 77.050 epochs with classical training. After the same number of epochs, using the newly proposed method, we have obtained a classification error of 2.79.

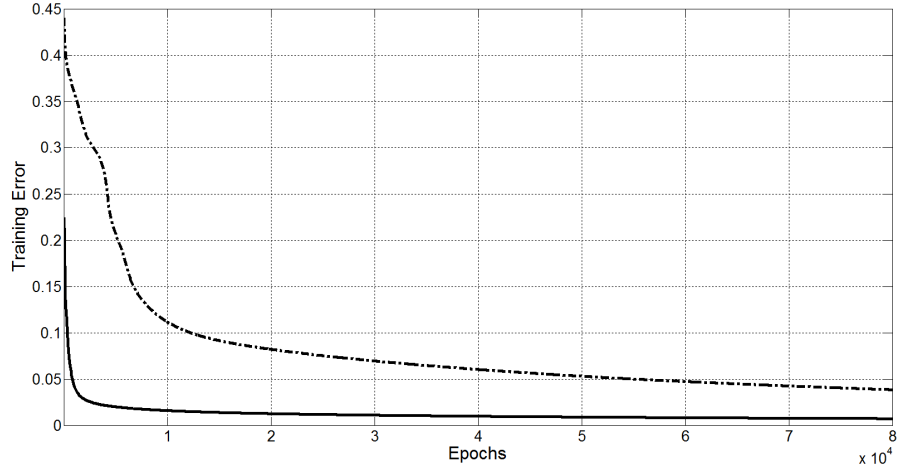


Figure 8: Training error for experiment 3. The newly proposed method (continuous line) shows a smaller error than the classical method (dash line) after 8×10^4 epochs. Also, the new method yields a faster convergence in terms of training error.

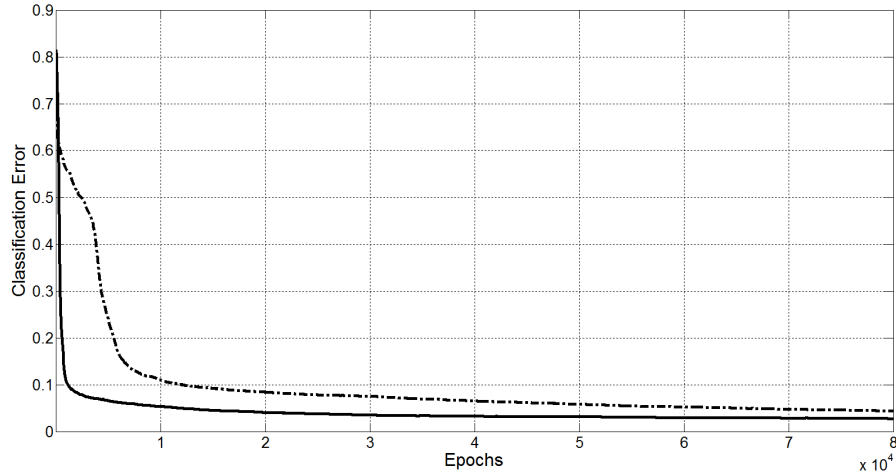


Figure 9: Classification error for experiment 3. The newly proposed method (continuous line) shows a better classification than the classical method (dash line). After 8×10^4 epochs, the classification error obtained with the new method is 2.76 vs. 4.39, which is the classification error obtained using the classical method.

4 Conclusions

We have presented a new method for training a feedforward neural network. The core of the method relies on the contraction property of some activation functions (e.g. sigmoid function) and the geometry underlying the training of FNN. As a result, we have obtained a new cost function that needs to be minimized during the training process. The main advantage of the new functional resides in the fact that we have less non-linearities introduced by activation functions.

The experiments that we have conducted show that our method leads to a faster learning convergence and also to a better recognition rate (classification error). In the first experiment, after 47.750 epochs, the classical method gives a classification error of 3.05 versus our method, which renders a classification error of 2.59. This behaviour remains valid for the next two experiments where we classification error for classical method was 2.97 versus 2.00 obtained with the new method, respectively 4.5 classification error obtained with classical method versus 2.79 the classification error obtained with the new method.

References

- [1] Ba, J. and Frey, B. (2013). Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3084–3092.
- [2] Baldi, P. and Sadowski, P. (2014). The dropout learning algorithm. *Artificial Intelligence*, 210C:78–122.
- [3] Broyden, C. G. et al. (1965). A class of methods for solving nonlinear simultaneous equations. *Math. Comp*, 19(92):577–593.
- [4] Dahl, G. E., Sainath, T. N., and Hinton, G. E. (2013). Improving deep neural networks for LVCSR using rectified linear units and dropout. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8609–8613. IEEE.
- [5] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning*, 12:2121–2159.
- [6] Fahlman, S. E. (1988). An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, Carnegie-Mellon Univ.

- [7] Fletcher, R. and Powell, M. J. (1963). A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163–168.
- [8] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier networks. In *AISTATS*, volume 15, pages 315–323.
- [9] Goldfarb, D. (1970). A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26.
- [10] Hanson, S. J. and Pratt, L. Y. (1989). Comparing biases for minimal network construction with back-propagation. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems (NIPS) 1*, pages 177–185. San Mateo, CA: Morgan Kaufmann.
- [11] Hestenes, M. R. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–436.
- [12] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. Technical Report arXiv:1207.0580.
- [13] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS 2012)*, page 4.
- [14] Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In Lippman, D. S., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 4*, pages 950–957. Morgan Kaufmann.
- [15] Levenberg, K. (1944). A method for the solution of certain problems in least squares. *Quarterly of applied mathematics*, 2:164–168.
- [16] Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML)*.
- [17] Malik, J. and Perona, P. (1990). Preattentive texture discrimination with early vision mechanisms. *Journal of the Optical Society of America A*, 7(5):923–932.
- [18] Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2):431–441.
- [19] Martens, J. (2010). Deep learning via Hessian-free optimization. In Fürnkranz, J. and Joachims, T., editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 735–742, Haifa, Israel. Omnipress.
- [20] Møller, M. F. (1993). Exact calculation of the product of the Hessian matrix of feed-forward network error functions and a vector in $O(N)$ time. Technical Report PB-432, Computer Science Department, Aarhus University, Denmark.
- [21] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *International Conference on Machine Learning (ICML)*.
- [22] Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate of $1/k^2$. *Soviet Mathematics Doklady*, 27(2):372–376.
- [23] Pearlmutter, B. A. (1994). Fast exact multiplication by the Hessian. *Neural Computation*, 6:147–160.
- [24] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press.

- [25] Schaback, R. and Werner, H. (1992). *Numerische Mathematik*, volume 4. Springer.
- [26] Schaul, T., Zhang, S., and LeCun, Y. (2013). No more pesky learning rates. In *Proc. 30th International Conference on Machine Learning (ICML)*.
- [27] Schraudolph, N. N. and Graepel, T. (2002). Conjugate directions for stochastic gradient descent. In Dorronsoro, J. R., editor, *Proc. Intl. Conf. Artificial Neural Networks (ICANN)*, volume 2415 of *Lecture Notes in Computer Science*, pages 1351–1356, Madrid, Spain. Springer Verlag, Berlin.
- [28] Shanno, D. F. (1970). Conditioning of quasi-Newton methods for function minimization. *Mathematics of computation*, 24(111):647–656.
- [29] Tieleman, T. and Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.
- [30] Weigend, A. S., Rumelhart, D. E., and Huberman, B. A. (1991). Generalization by weight-elimination with application to forecasting. In Lippmann, R. P., Moody, J. E., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems (NIPS) 3*, pages 875–882. San Mateo, CA: Morgan Kaufmann.
- [31] West, A. H. L. and Saad, D. (1995). Adaptive back-propagation in on-line learning of multilayer networks. In *NIPS*, pages 323–329.
- [32] Y. LeCun, L. Bottou, Y. B. and Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- [33] Zeiler, M. D. (2012). ADADELTA: An Adaptive Learning Rate Method. *CoRR*, abs/1212.5701.